Attorney Docket No.002-26: CSN 7523

PATENT APPLICATION

BACKWARD-COMPATIBLE PARALLEL DDR BUS FOR USE IN HOST-DAUGHTERCARD INTERFACE

Inventors: JAMES EVERETT GRISHAW, a citizen of the United States of America, residing at 3433 Wheeling Drive, Santa Clara, California 95051;

MICKEY RAMAL HENNIGER, a citizen of the United States of America, residing at 332 Boulder Drive, Buchanan Dam, Texas 78609.

Assignee: CISCO TECHNOLOGY, INC. (A California Corporation)

Entity: LARGE

BACKGROUND OF THE INVENTION

[01] Incompatibility between new and existing products is a major problem in many technical disciplines including networking and signal switching. Often new products are developed having increased data rates but customers have invested heavily in legacy products operating at a slower data rate.

[02] For example, many routing platforms have slots for WAN (Wide Area Network) Interface Cards (WICs) that provide interfaces between the routing platform and wide area networks.

[03] The legacy WICs were conceived several years ago, and originally targeted PHY devices with a maximum bit rate of a few Mbps. Since then, the explosion of the internet and broadband availability has enormously increased the speed and complexity required of Cisco router interfaces. There is a need to expand the legacy WIC functionality to enable existing routing platforms to continue to meet present as well as future interface card requirements.

BRIEF SUMMARY OF THE INVENTION

[04] In a first embodiment of the invention, pins of a legacy host-daughtercard interface are appropriated to define a high-speed parallel bus in a new high-speed interface that is backward compatible with the legacy interface.

[05] In another embodiment of the invention, data and control frames are transmitted over the high-speed parallel bus where a control line is asserted to distinguish between data and control frames.

[06] In another embodiment of the invention, the pins appropriated functioned as a parallel port in the legacy interface and the host utilizes read and write frames to perform the function of a legacy parallel port in the high-speed interface.

[07] In another embodiment of the invention, an interrupt control frame is utilized by the daughtercard to interrupt a processor on the host platform.

[08] In another embodiment of the invention, address extension bits are held in a register to extend the buffer addresses held in a descriptor in host memory.

[09] In another embodiment of the invention, a cookie on the daughtercard indicates whether the daughtercard supports the legacy interface or the new high-speed interface. The host reads the cookie to determine which interface is supported.

[10] Other features and advantages of the invention will be apparent from the following detailed description and appended drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[11] Fig. 1 is a block diagram of a legacy Host/WIC interface;

[12] Fig. 2 is a block diagram of an embodiment of the parallel DDR bus;

[13] Fig. 3 is a high-level block diagram of an embodiment of the host-WIC system;

[14] Fig. 4 is a more detailed diagram of the HWIC interface to the HWIC connector;

[15] Fig. 5 is a flow chart of an embodiment of the steps for distinguishing between a legacy and HWIC interface;

[16] Figs. 6A and B are diagrams depicting the format of the 8-bit HWIC read and read response commands;

[17] Figs. 7A and B are diagrams depicting the format of the 8-bit HWIC write and write response commands;

[18] Fig. 8 is a diagram depicting the format of the interrupt control frame;

[19] Fig. 9 is a diagram depicting the format of the Generic data frame;

[20] Fig. 10 is a diagram depicting the format of the HWIC-to-host DMA Read Request Frame;

[21] Fig. 11 is a diagram depicting the format of the Host-to-HWIC DMA Read Response; and

[22] Fig. 12 is a diagram depicting the HWIC-to-host DMA Write Request.

DETAILED DESCRIPTION OF THE INVENTION

[23] Reference will now be made in detail to various embodiments of the invention. Examples of these embodiments are illustrated in the accompanying drawings. While the invention will be described in conjunction with these embodiments, it will be understood that it is not intended to limit the invention to any embodiment. On the contrary, it is intended to cover alternatives, modifications, and equivalents as may be included within the spirit and scope of the invention as defined by the appended claims. In the following

description, numerous specific details are set forth in order to provide a thorough understanding of the various embodiments. However, the present invention may be practiced without some or all of these specific details. In other instances, well known process operations have not been described in detail in order not to unnecessarily obscure the present invention.

[24] The invention will now be described with reference to various embodiments implemented in a routing platform. In the following, the term routing platform is utilized broadly to include any component such a router, bridge, switch, layer 2 or layer 3 switch, gateway, etc., that refers to components utilized to implement connectivity within a network or between networks. In the following, embodiments will be described, by way of example, not limitation, that operate on routing platforms designed and manufactured by the assignee of the present patent application. However, it is understood by persons of skill in the art that the invention has broad utility in any routing platform.

[25] A brief overview of an embodiment of the invention will now be described. In the following, for ease of description, the current embodiment will be denoted the High-speed WAN Interface card (HWIC).

[26] The legacy WIC/Host interface includes a number of serial interfaces and a parallel interface. In the presently described embodiment of the HWIC the physical interface is kept the same as the legacy WIC/Host interface so the legacy VWIC modules are pin compatible and may be inserted into host platforms built for the new HWIC interface. In order to maintain compatibility with existing VWICs, i.e., VICs (Voice Interface Cards) and WICs, this embodiment of the HWIC contains the WIC/VIC Serial Peripheral Interface (SPI) and may contain some combination of WIC/VIC Serial Interfaces. For those serial interfaces, the interface requirements, pin definitions, interface operations, and interface timings are identical to those of the legacy WIC/VIC.

[27] This compatibility is further achieved by appropriating pins of the legacy WIC/Host interface for new functions. The detailed appropriation of pins for a specific interface, designed and manufactured by the assignee of the present application, will now be described by way of example not limitation.

[28] The legacy WIC/Host parallel interface is depicted in Fig. 1 and is a generic 8-bit parallel bus which functions as a parallel port to allow the host to access the legacy WIC's on-board registers as well as other network devices. This legacy WIC/Host interface includes a byte oriented bus, connecting a host 10 and WIC 11, with which the host must be able to access any legacy WIC's address located in the odd or even byte boundary.

[29] The signals depicted in Fig. 1 are: RD_L: Read signal; WR_L: Write signal; CS_L: enable signal; Not_RDY: not ready signal which is asserted (LOW) when the data is not ready; ADDR[7:0]: Address bus; and DATA[7:0]: Data bus.

[30] In addition to the legacy interfaces above, the HWIC of the presently described embodiment supports an 8-bit double-data rate (DDR) bi-directional bus depicted in Fig. 2. This DDR bus is a synchronous bus. This interface will be used to: 1) provide a high-speed data path between the host platform and the HWIC for WAN data; 2) provide access to the module's on-board registers; and, 3) to provide a DMA path for HWIC on-board devices.

[31] In this embodiment, TxClk and RxClk run at 25 MHz. The buses operate in Double Date Rate (DDR) mode, where data is captured on every clock edge. This provides aggregate bandwidth of 800 Mbps (400 Mbps in each direction).

[32] The signals depicted in Fig. 2 are: TxD[7:0]: Transmit Data bus from host; TxCtrl: Transmit Control bit from host; TxClk: Transmit Clock from host; RxD[7:0]: Receive Data bus to host; RxCtrl: Receive Control bit to host; and, RxClk: Receive Clock to host.

[33] In this embodiment, the ADDR[7:0] (Address pins) of the legacy Host/WIC parallel interface have been appropriated as the TxD[7:0] (Transmit Data bus from host) in the DDR bus of the Host/HWIC interface. Also, the data pins of the DATA[7:0] (Data bus) of the legacy Host/WIC parallel interface have been appropriated as the RxD[7:0] (Receive Data bus to host) in the DDR bus of the Host/HWIC interface. Additionally, the CS_L (enable signal) of the legacy interface has been appropriated as the TxCrl pin of the Host/HWIC interface and the RD_L pin of the legacy Host/WIC interface has been appropriated as the RxCtrl pin of the Host/HWIC interface. Further, the legacy Host/WIC parallel interface echo clock pins have been appropriated as the TxClk and RxClk pins in the DDR bus of the Host/HWIC interface.

[34] As described above, the new HWIC interface is plug compatible with the legacy WIC/Host interface. However, the functionality of some of the pins will differ in the new implementation.

[35] Note that, in the presently-described embodiment, only the parallel port pins and the "Echo Clock" (TxCE) pins of the legacy Host/WIC parallel interface are cannibalized for the HWIC high-speed bus. This provides for backwards compatibility by still leaving the SPI lines, SCC's (Serial Communication Controllers), interrupt lines, etc available for their traditional uses in the legacy Host/WIC parallel interface.

[36] Fig. 3 is a high-level block diagram of the host-WIC system. In Fig. 3 the host 10 includes host memory 14 and a CPU 15 coupled to a host termination logic block 16 including an HWIC interface 18. The HWIC 12 includes an HWIC terminating logic block 20 including a an HWIC interface 22. The host HWIC interface 22 is coupled to a 68-pin connector 24, which as described above, is pin compatible with the legacy HWIC connector.

[37] Fig. 4 is a more detailed diagram of the HWIC interface 22 coupled to the HWIC connector 24. The principal components from the HWIC side are the Serial Peripheral Bus (SPI) 40 connected to a cookie 42, in this embodiment a non-volatile memory, such as an non-volatile RAM (NVRAM) in the form of an EEPROM, storing information about a particular implementation, the HWIC DDR parallel bus coupled to the HWIC termination logic 16, in this embodiment an FPGA including a set of configurable registers 44, and a power pin (not shown) for supplying power to the HWIC. In this embodiment, the FPGA includes registers are utilized for configuring the HWIC.

[38] An important component of backward compatibility is providing the host with a system for determining whether a legacy or upgraded part has been inserted in the slot. In the presently described embodiment this function is required because, although the parts are pin compatible, certain pins are used to perform completely different functions. The system for implementing card identification and preventing problems due to incompatible signals will now be described.

[39] In this embodiment, an HWIC can be plugged into an older host platform, where the old host platform can still access the cookie on the HWIC via the SPI lines and find out that this inserted WIC is not supported. The HWIC will be required to not drive the Rx lines until "enabled", so that the WIC parallel port bus pins will not cause bus contention in the event that an HWIC is inserted into an older host platform. Also, by design the legacy SCC's are still available on an HWIC and can be used for purposes such as management channels.

[40] The procedure for powering up the HWIC of the currently described embodiment and facilitating compatibility with the legacy Host/WIC parallel interface will now be described in detail with reference to the flow chart of Fig. 5.

1. Power is applied to the host platform, and the host CPU boots up. Alternatively, the host CPU issues a hard reset to the HWIC (in response to "clear interface" or some other reason to start over from scratch).

2. If the HWIC has an on-board configuration ROM for its FPGA, the HWIC FPGA is automatically (re)configured. However, the HWIC continues to tri-state the legacy parallel port pins: Addr[7:0], Data[7:0], nCS, nRD, nWR, nRDY.
3. The host CPU uses the SPI to read the NVRAM on the HWIC and identify it. If the platform is older or otherwise does not support this type of HWIC, a message is displayed to the user and execution stops here.
4. If the HWIC does NOT have an on-board configuration ROM for its FPGA, the host processor downloads the configuration bit stream to the HWIC FPGA by use of the 8-bit parallel port or SPI. Again after download, the HWIC FPGA is tri-stating all of its parallel-port lines (enumerated in (2) above).
5. The host CPU tri-states the RD_L pin to avoid bus contention when the HWIC's DDR mode is enabled.
6. The host CPU turns off (legacy WIC) parallel port mode for this WIC interface and enables HWIC DDR mode in the host termination logic.
7. The host CPU enables HWIC termination logic to drive its 10 HWIC-to-host DDR pins (RxData[7:0], RxClk, and RxCtrl) by writing to a special register in the HWIC termination logic. Note that although the HWIC-to-host DDR pins are tri-stated, the host-to-HWIC pins (TxData[7:0], TxClk, and TxCtrl) are still active so that commands can be transmitted from the host to the HWIC.
8. The host verifies that the HWIC FPGA has been properly downloaded and that the DDR bus is operational by reading an HWIC register through the DDR bus.
9. HWIC configuration continues by means of the HWIC DDR bus.

[41] The operation of the DDR bus itself will now be described in more detail. Two types of frames are defined: control frames and data frames. Data frames are used to pass the larger packets of data between the host and the HWIC, en route to and from the line interface. The control frames are smaller in nature, and, since they perform administrative functions, they pre-empt the data frames in order to reduce latency. There is also another important difference between the two types of frames – the control frame formats are generated and received in hardware, whereas the data frame format is determined by the host processor (with the exception of the DMA frames noted below).

[42] The control bits (RxCtrl, TxCtrl) distinguish data frames from control frames: TxCtrl, RxCtrl = "0": indicates that streaming data is being passed, TxCtrl, RxCtrl = "1": indicates that control information is being passed. In the present embodiment, the Tx and Rx buses are always transmitting bytes: either control bytes, data bytes, or idle bytes.

[43] CONTROL FRAMES

[44] The functions and implementations of control frames in the presently described embodiment will now be described. The control frames have three principal functions: 1) flow control by means of stop, resume, and transmit control characters, 2) read/write commands utilized to perform the functions of the legacy parallel port, and, 3) interrupt frames. Each will be described in more detail below.

[45] Turning first to the control frame format, A control frame (or byte) is transmitted if and only if the respective control bit (TxCtrl or RxCtrl is "1"). A data frame is transmitted if and only if the respective control bit (TxCtrl or RxCtrl) is "0". Hence for example a 0xFF transmitted with the control bit set to 1 has a specific meaning (start/end of data frame); whereas a 0xFF transmitted with the control bit set to 0 is to be interpreted as data.

[46] The control Frame characters for implementing flow control will now be set forth in detail.

[47] 0xFF: start / end of data frame or IDLE between data frames. Each data frame must begin with the 0xFF control byte, and does not end until the 0xFF control byte is received. Flag sharing (two data frames separated by one 0xFF flag) is allowed.

[48] 0xF0: IDLE byte in middle of data frame. This control character can be inserted into the middle of the frame in order to indicate an "IDLE" condition. The receiver will do nothing with this byte other than discard it. The purpose of the IDLE byte is so that the data frames will NEVER underrun on the host/HWIC interface.

[49] 0xFA: ABORT. If this control character is encountered, any data frame in progress is immediately terminated, and an "abort" is generated. At least one 0xFF flag must be sent following this to identify the beginning of the next data frame. The ABORT character is required primarily to signal a prematurely terminated the Generic Data Frame. Without the ABORT character it would be difficult to determine a premature termination because the Generic Data Frame does not contain a Length field. However, the ABORT character may also be used to signal premature termination of a DMA Data Frame.

[50] 0xF3: Suspend Transmit Request. This control character can be sent either by the host or by the HWIC, to request that the other party suspend transmitting data frames. This is intended for flow control purposes, to prevent the overflow of a FIFO that is becoming full. Upon receiving a "suspend transmit" request, the transmitting party may send a maximum of 32 more bytes before ceasing transmission of all "data frame" bytes.

[51] 0xFC: Resume Transmit Request. This control character can be sent either by the host or by the HWIC, to request that the other party resume data frame transmission, after transmission has been suspended by the "Suspend Transmit Request" control byte.

[52] 0xC0: Low Water Mark. This control character is sent from the Host Termination Logic to the HWIC when the Receive Queue Count becomes less than or equal to the Receive Queue Low Water Mark register value and the Water Mark control bytes

are enabled with Configuration Register bit 7. Once a Low Water Mark control byte is sent, no more Low Water Mark control bytes will be sent until a High Water Mark control byte is sent first.

[53] 0xCF: High Water Mark. This control character is sent from the Host Termination Logic to the HWIC when the Receive Queue Count becomes greater than or equal to the Receive Queue High Water Mark register value and the Water Mark control bytes are enabled with Configuration Register bit 7. Once a High Water Mark control byte is sent, no more High Water Mark control bytes will be sent until a Low Water Mark control byte is sent first.

[54] In this embodiment, the HWIC response to the Low Water Mark / High Water Mark control bytes is design-specific. Typical uses might be to enable selective packet dropping on the HWIC (e.g. drop data packets but continue to send control packets if the Receive Buffer Ring is getting full) or as a warning to allow the HWIC to apply flow control to the transmitter on the other side of the line.

[55] The above single-byte control characters may ONLY be inserted into data frames. The 0xF0 IDLE byte cannot be inserted into a control frame, nor do control frames require the 0xFF delimiters (since control frames are of predetermined length).

[56] The read write control commands will now be described in more detail. The legacy WIC parallel port pins have been cannibalized for the HWIC DDR data buses. The read/write control frames replace the lost functionality of the legacy parallel port. The read and write control commands are utilized by the host to read register data from the HWIC over the DDR bus.

[57] In order to maintain the low latency of the legacy parallel port operations, these control frames will immediately be inserted (on a byte boundary) into any data frame that is currently in progress, and the HWIC termination logic will immediately respond, again pre-empting any data frame that may be in progress.

[58] The format of the HWIC read and read response commands for transferring 8 bits of data are depicted in Figs. 6A and 6B. The read command includes an opcode identifying the type of command, e.g. 8-bit, 16-bit, or 32-bit read, an address, and CRC bits as depicted in Fig. 6A. The read response command includes an opcode identifying the type of command, e.g. 8-bit, 16-bit, or 32-bit read response, address, read data, and CRC bits.

[59] The HWIC read is sent from the host to the HWIC in order to query an indicated memory location. The HWIC will respond with the "8-bit HWIC read response"

frame. Upon receiving the "HWIC read" command, the HWIC termination logic will read the HWIC memory location at "Address" and send the resulting data to the host framed as shown in the Fig. 6B.

[60] The format of the HWIC write and write response commands are depicted in Fig. 7A and B. The write command, depicted in Fig. 7A, includes an opcode identifying the type of command, e.g. 8-bit, 16-bit, or 32-bit write, and address, data to be written, and CRC bits. The write response, depicted in Fig. 7B, includes an opcode.

[61] The HWIC write command is sent from the host to the HWIC in order to write the transmitted data to the desired HWIC memory location. After the HWIC termination logic has successfully completed the write, it will send this byte back to the host in order signal completion. The Host Termination Logic should not initiate the next HWIC read or write command until it has received HWIC write response control byte from the HWIC.

[62] The interrupt control frame will now be described. The format of the interrupt control frame is depicted in Fig. 8. The HWIC interrupt frame is sent from the HWIC to the host upon detection of some interrupt event that is unmasked. The 16 "source" field bits correspond to 16 bits in the HWIC termination logic Interrupt Event Registers. Upon receipt of the HWIC Interrupt frame, the "Source" field will be logically ORed with these Event Register bits to create the new event register bit values. This may in turn interrupt the host processor (if the corresponding event register bit is unmasked).

[63] The use of the HWIC interrupt frame reduces interrupt latencies by eliminating the need for the host to poll the HWIC registers to determine the interrupt source on the HWIC. The HWIC interrupt frame may also be used to alleviate any temptation to have the host CPU poll an HWIC register while waiting for an event to occur (this could severely degrade performance).

[64] Some host platforms support 3 levels of HWIC interrupts to the host – Error interrupt, Management / Config interrupt, and Network interrupt. In order to support hosts that support these three interrupt types, the HWIC Interrupt Frame bits (bits 15-0 of the Source field) are partitioned as follows: Bits 15 – 12 : Error interrupts, Bits 11 – 8 : Management / Config interrupts, Bits 7 – 0 : Network interrupts. Each interrupt group is mapped to a separate Interrupt Enable and Interrupt Event register.

[65] DATA FRAMES

[66] There are two types of data frames: DMA data frames and Generic data frames. They have very different uses.

[67] The following are some of the similarities between DMA and Generic data frames: (1) By definition, a data frame is transmitted if and only if the respective control bit (TxCtrl or RxCtrl) is "0"; (2) All data frames carry a CRC8 as the last byte; (3) All data frames start with an encapsulation header. All Generic data frames start with the 0x11 byte. All DMA data frames begin with 0x61, 0x62, 0x66, or 0x67.

[68] The DMA and Generic data frames have different purposes. On the one hand, the DMA data frame serves as a vehicle for HWIC-initiated DMA transactions. The DMA data frame allows the HWIC to read and write to host memory. DMA data frames are processed entirely in hardware, so it is not necessary for the host processor to be involved in these transactions. For example, simple register accesses are inefficient for reading blocks of registers so DMA frames are utilized to transfer blocks or register data from the HWIC to the host over the DDR bus. The HWIC requires data structures set up in host processor memory to support its Generic Data Frame transmit and receive operations. All of these data structures are shared by the host processor and the host termination logic through DMA accesses.

[69] On the other hand, the Generic data frame will be an implementation-specific frame agreed upon between the host driver software and the HWIC termination logic. For example, a data frame may simply be an IP packet or ATM cell that the HWIC termination logic sends to or receives from the PHY device on the HWIC. Or there may be an encapsulation, such as a header with an 8-bit port number indicating which PHY device the HWIC termination logic sends/receives the packet from. The purpose of the Generic data frame is to allow the HWIC designer the flexibility to create whatever frame format will best suit the specific HWIC being designed.

[70] The DMA and Generic data frames are also processed differently. On the one hand, the DMA data frames originate in hardware (on the HWIC if it is a DMA request, or on the host if it is a DMA response). Upon receipt they are also processed entirely in hardware, leaving the host CPU completely uninvolved in the transaction (until perhaps the very end, after the transaction is completed, when the host is notified via interrupt that a DMA transaction has occurred).

[71] On the other hand, the Generic data frame is processed almost entirely in software on the host side. Any special encapsulations for transmit frames (outside of the initial 0x11 byte) must be created by the host processor. And likewise received frames are also entirely processed by the host processor (after the leading 0x11 byte is removed).

[72] On the HWIC side, by necessity (unless a processor resides on the HWIC) all Generic data frames are processed in hardware, which has carnal knowledge of the HWIC-specific Generic data frame format, that has been agreed upon between the host CPU and the HWIC hardware.

[73] The flexibility of the types of data frames is different. Since the DMA data frame is processed entirely in hardware, it is not flexible and must remain exactly the same format (as described below) from HWIC to HWIC.

[74] Since the Generic data frame is created and parsed by the host processor, the format of the Generic data frame is extremely flexible. It is intended that the host CPU will choose Generic data frame format that will facilitate the design of each particular HWIC.

[75] **GENERIC DATA FRAMES**

[76] The format of the Generic data frame will now be described with reference to Fig. 9.

[77] Command Opcode: Generic data frames passed between the HWIC and the host must begin with a 0x11 byte in order to indicate Generic data frame encapsulation.

[78] Data: The "Data" field may be any data of any length (providing of course that it follows the implementation-specific format agreed upon by the host driver software and the HWIC termination logic). For example, this could be an IP packet, ATM cell, or PPP frame, encapsulated with a port number or VC number in the header.

[79] (Optional) Rx Flags: For Generic Data Frames passed from the HWIC to the host platform, the upper 2 bits of the Rx Flags byte are written to the Receive Buffer Descriptor word 1 bits 23-22. This is to allow the convenient passing of error / status information that may not be readily available for insertion into the beginning of the Generic Data Frame, for example line CRC calculations that are not completed until the end of the frame arriving at the PHY, on HWICs that do not store the entire frame before passing it up to the host.

[80] The Optional Rx Flags byte is also placed in to the receive buffer and counted in the Data Length field of the Receive Buffer Descriptor, so if the Rx Flags functionality is not needed this byte may be used for frame data as long as the host processor ignores the Rx Flags in the Receive Buffer Descriptor (word 1 bits 23-22). This byte does not provide any special functionality for the transmit direction, and should be used for valid frame data.

[81] CRC8: 8-bit CRC calculated on all frame bytes except the CRC8 field itself. Additionally, the CRC8 is NOT calculated over any inserted control frames.

[82] **DMA FRAMES**

[83] The DMA frames will now be described. There are three types of DMA transactions: HWIC-to-host DMA Read Request, Host-to-HWIC DMA Read Response, and HWIC-to-host DMA Write Request. All DMA transactions are initiated by the host. Each of these frame formats will now be described.

[84] The format of the HWIC-to-host DMA Read Request Frame is depicted in Fig. 10.

[85] DMA Read Opcode: This byte should be set to 0x61 to indicate that the HWIC is requesting a DMA read from the host.

[86] Address: 32-bit start address for the DMA operation in host memory space.

[87] Length: 16-bit length in bytes of the requested DMA read – how much memory the host should read and send back to the HWIC.

[88] CRC8: 8-bit CRC calculated on all frame bytes except the CRC8 field itself. Additionally, the CRC8 is NOT calculated over any inserted control frames.

[89] The Host-to-HWIC DMA Read Response Frame will now be described with reference to Fig. 11. This frame is sent from the host to the HWIC in response to the "HWIC-to-host DMA Read Request" frame.

[90] DMA Read Response Opcode: This byte should be set to 0x62 to indicate that this is a "host-to-HWIC DMA Read Response" frame.

[91] Address: 32-bit start address for the DMA operation in host memory space.

[92] Length: 16-bit number of bytes to follow in the "data" field.

[93] Header CRC8: 8-bit CRC calculated on all previous frame bytes except the "Header CRC8" field itself.

[94] Data: Data bytes from host memory.

[95] Frame CRC8: 8-bit CRC calculated on all frame bytes except the CRC8 field itself. Additionally, the CRC8 is NOT calculated over any inserted control frames.

[96] The HWIC-to-host DMA Write Request Frame will now be described with reference to Fig. 12.

[97] DMA Write Opcode: This byte should be set to 0x66 or 0x67 to indicate that the HWIC is requesting a DMA write to the host.

[98] 0x66 = Do not interrupt the host after this DMA transaction is completed

[99] 0x67 = DO interrupt the host after this DMA transaction is completed.

[100] If opcode 0x67 is used, the host termination logic will interrupt the host CPU after all the data has been received and sent to host memory or the host memory controller. Note that this does not necessarily guarantee the data will be completely written to host memory before the host CPU is interrupted; however, in this event it is the intention of opcode 0x67 to minimize the latency between the interrupt and the memory write completion as much as can reasonably be achieved. This will enhance system performance by minimizing instances that the CPU must abort and re-try later (when it receives the 0x67 interrupt but then discovers that the required data is not in memory yet).

[101] Address: 32-bit start address for the DMA operation in host memory space. Note: The host termination logic will implement an address mask function to protect various sections of host memory from accidental/erroneous HWIC writes.

[102] Length: 16-bit number of bytes to follow in the "data" field.

[103] Header CRC8: 8-bit CRC calculated on all previous frame bytes except the "Header CRC8" field itself.

[104] Data: Data bytes to write to host memory.

[105] Frame CRC8: 8-bit CRC calculated on all frame bytes (i.e. starting with "Command Opcode") except the "Frame CRC8" field itself.

[106] **Buffer Address Extension**

[107] As described above, several host memory data structures are associated with HWIC and all of these data structures are shared by the host processor and host termination logic through DMA accesses.

[108] Transmit and Receive Buffer Descriptors are host data structures that include a 32-bit Buffer Address in host memory of the buffer which contains the data to be transferred using Generic Data Frames.

[109] This embodiment includes Transmit and Receive Buffer Address Extension registers that allow extension of the Buffer Address held in a descriptor by an additional 8 bits. If the host termination logic does not support more than 32 bits of host address space this register holds a value of zero.

[110] If address extension is supported then the HWIC termination logic uses the 8 transmit buffer extension bits to extend the Buffer address to a 40-bit address as shown here:

[111] {"Transmit Buffer Extension (8 bits)", "Transmit Buffer Address (32 bits)"} = {"Extended Address" (40 bits)}

[112] Components of the invention may be implemented as program code, stored on a computer readable medium, that is executed by a digital computer. The computer readable medium may include, among other things, magnetic media, optical media, electro-magnetic fields encoding digital information, and so on.

[113] The invention has now been described with reference to the preferred embodiments. Alternatives and substitutions will now be apparent to persons of skill in the art. For example, the formats of the various frames are described by way of example and other frame formats can be utilized as is known in the art. Further, the pins of the legacy interface that are appropriated for the high- speed interface will vary depending on the pin functions of the legacy interface. Accordingly, it is not intended to limit the invention except as provided by the appended claims.